

**Vacation Bursary Student & Project Details Report**

|  |  |
| --- | --- |
| **Student’s full name** | James David King |
| **Email Address** | James.king3@durham.ac.uk |
| **Gender** | Male |
| **Age** | 20 |
| **Country of Origin** | England |
| **Ethnicity** | White British |
| **EPSRC Theme** | ICT |
| **Start date of the undergraduate degree course** | October 2011 |
| **End date of the Undergraduate Degree Course** | June 2014 |
| **Undergraduate Degree Course Details** | Computer Science - G400 |

**Vacation Bursary Project Details**

|  |  |
| --- | --- |
| **Start Date of the vacation Bursary** | 8th of July 2013 |
| **End Date of the Vacation Bursary** | 9th of September 2013 |
| **Vacation Bursary Project Title** | Building a Modern Oberon-2 Compiler using the LLVM Compiler Infrastructure |

|  |  |
| --- | --- |
| **Objectives and how this fits into a larger scale project, (if relevant)** | 1. Implement a working parser for the Oberon-2 language, either using recursive parsing or scanner/parser generator tools such as flex/bison. 2. Connect the Oberon-2 parser to the LLVM back-end and generate working code for simple programs. 3. Extend the Oberon-2 language to provide interfaces to system libraries, i.e. in order to provide access to most POSIX-type functionality. This somewhat technical goal is a key aspect of making the compiler usable to a wide audience. 4. Extend the language with basic types for SIMD-vectors to allow for the straight-forward use of explicit vectorization, which is missing from all current higher-level programming languages. 5. Extend the language to support the LLVM PTX, i.e. CUDA-compatible GPU, back-end. (Non-essential) |
| **Success Story (if relevant)** | For the first objective, I experimented with writing my own parser generator. This was partially due to my curiosity of how one would work, and also to try and keep external dependencies for the project limited to only LLVM. An extended Backus-Naur Form (EBNF) file describing the syntax of Oberon-2 was part of the official specification of the language, so the parser generator was designed to read this directly as input (with some small modifications to improve the parsed structure of the file). Within a couple of days I had a parser that would be able to recognise if a given file contained syntactically correct Oberon-2 code, and would pinpoint the exact location of the first syntax error found if the file was invalid. If the file is valid, a syntax tree matching the format of the file using the rules specified in the EBNF document is automatically generated. The parser generator itself has nothing tying it to the language, and could be used in the future for parsing other languages by providing a corresponding EBNF file. After the completion of the project I will probably spend some time cleaning up the generator and publishing it as a separate project.  While a source code file may be syntactically correct, type errors or references to undeclared symbols would render it semantically invalid. To check if this is the case, the nodes in the syntax tree are swapped out for context sensitive nodes that perform checks on their contents to ensure no semantic errors exist. The system is designed to find every existing error, not just the first. These errors are all listed in the command line output in a format that shows exactly where in the source file they occurred, along with a code snippet formatted to show the symbols that generated the problem. I feel the error reporting used in this project is more intuitive than most command line compilers I have experienced in the past, perhaps making this a suitable compiler for a programming course where students understanding the cause of errors in their code is essential.  Having finished the parser portion of the project early, I was able to get a head start generating the intermediary representation (IR) used by LLVM. This was the most difficult portion of the project, but became easier as my familiarity with the IR increased. Around this time, my supervisor managed to contact the original designer of Oberon and Oberon-2, Niklaus Wirth, who seemed interested in the project. At some point in the coming weeks we plan to send our proposed additions to the language to Professor Wirth to get some feedback, as we would like to adhere to his design principles where possible. External library interfaces proved to be simpler than expected to implement, and a working clone of the game Pong using OpenGL was entirely written in Oberon-2 with the new external symbol referencing syntax. SIMD operations have also been added successfully ahead of schedule, so I will spend the remaining time preparing the project for release. The initial proposal included a final non-essential objective to attempt to support LLVM PTX, but my supervisor and I both agree that time would be better spent on writing documentation for the project. |